

### Ejemplo práctico de carga de datos con PostgreSQL

**NOTA:** Para seguir los pasos descritos en el presente documento, es necesario el fichero *CodigosPostales.txt*

El presente ejemplo de carga de datos parte de la premisa de tener instalado el Sistema Gestor de Bases de Datos (SGBD) PostgreSQL (7.4 o superior) y de disponer de un editor de texto de tipos *ASCII* para la manipulación del fichero de texto de carga.

El ejemplo de carga de datos nos servirá para introducir en una tabla 10.144 códigos postales de España, con los correspondientes nombres de ciudades.

Los pasos de que consta la carga son:

- Análisis del fichero ASCII y de los campos a cargar.
- Creación de los campos.
- Edición del fichero fuente.
- Carga de datos.
- Consultas de prueba.

#### Análisis del fichero de carga y de los campos a cargar.

Editando el fichero con el editor para ficheros ASCII, se deberá proceder a determinar si existe una primera fila con la definición de las columnas, el tipo de separación entre los campos, si el salto de líneas es el habitual, y si existe una última línea vacía al final del fichero.

En el caso de estudio observamos que existe una fila de cabecera que nos facilitará la creación de las columnas con sus atributos, y que se debe de insertar una última fila vacía a final de fichero.

La primera línea del fichero es:

```
CPRO CMUN Nombre_Municipio CP Municipio_CP Lugar_CP
```

Observando los datos comprobamos que:

CPRO: Corresponde a la numeración de las provincias.

CMUN: Corresponde al código de municipio.

Nombre\_Municipio: Tal como su nombre indica.

CP: Son los 5 dígitos del código postal.

Municipio\_CP: Municipio verificado para el CP.

Lugar\_CP: Referencia adicional de ubicación.

Para realizar la asignación de los tipos de datos de cada columna, recordaremos que únicamente se deben asignar los tipos numéricos a las columnas que contengan datos con los que se deban realizar operaciones aritméticas. En este caso, consideraremos que todas deben de ser de tipo carácter.

La anchura de las columnas vendrá determinada por la fila que tenga el valor máximo. Según se compruebe la homogeneidad, se optará por un tipos de anchura fija (CHAR) o por un tipo de columna de longitud variable (VARCHAR).

Durante el análisis de los datos optaremos por ajustar al máximo los tamaños de las columnas. En el caso de disponer de alguna herramienta para realizar comprobaciones previas a la carga

(según el volumen de datos), buscaríamos directamente la de mayor anchura para crear las columnas.

Según los datos que se observan en el editor de textos, optaremos por crear las columnas con los siguientes atributos:

```
CPRO: CHAR(2)
CMUN: CHAR(4)
Nombre_Municipio: VARCHAR(35)
CP: CHAR(5)
Municipio_CP: VARCHAR(35)
Lugar_CP: VARCHAR(35)
```

### Creación de los campos.

Como inicialmente no existe una columna (o combinación de ellas) que no tenga valores repetidos, optaremos por crear una adicional como Clave Principal (Primary key - PK). Los valores de la misma se insertaran automáticamente con un disparador (trigger) que obtendrá los valores de una secuencia o dispensador. La columna la llamaremos ID\_CP

La sentencia SQL para la creación de la tabla será:

```
CREATE TABLE CODIGOS_POSTALES (
    ID_CP decimal NOT NULL,
    CPRO CHAR(2) NOT NULL,
    CMUN CHAR(4) NOT NULL,
    Nombre_Municipio VARCHAR(45) NOT NULL,
    CP CHAR(5) NOT NULL,
    Municipio_CP VARCHAR(35),
    Lugar_CP VARCHAR(35),
    CONSTRAINT PK_ID_CP PRIMARY KEY (ID_CP));
```

La sentencia para la creación de la secuencia (dispensador) será:

```
CREATE SEQUENCE SQ_Inserta_CP START 1;
```

La sentencia para la creación del procedimiento que será llamado por el disparador, será: (es obligatorio crear primero el procedimiento y posteriormente el disparador)

```
CREATE FUNCTION FU_Inserta_CP() RETURNS TRIGGER AS '
BEGIN
    NEW.ID_CP := (SELECT NEXTVAL('SQ_INSERTA_CP'));
    RETURN NEW;
END;
' LANGUAGE 'plpgsql';
```

En el caso de que PostgreSQL indique error 42704 y que se debe de instalar el lenguaje PLPGSQL, se debe de ejecutar:

```
CREATE FUNCTION plpgsql_call_handler() RETURNS language_handler AS
'$libdir/plpgsql' LANGUAGE C;
```

```
CREATE TRUSTED PROCEDURAL LANGUAGE plpgsql
HANDLER plpgsql_call_handler;
```

La sentencia para la creación del disparador será:

```
CREATE TRIGGER TR_Inserta_CP
BEFORE INSERT ON CODIGOS_POSTALES
FOR EACH ROW
EXECUTE PROCEDURE FU_Inserta_CP();
```

En el caso de eliminar la tabla, el disparador y la función se borran automáticamente.

### Edición del fichero fuente.

Dado el tipo de separación de los datos del fichero fuente (tabuladores), no podremos indicar a la instrucción de carga (COPY) que no tenga en cuenta la línea de cabecera. Debido a ello, la eliminaremos desde el editor de textos. Adicionalmente, deberemos insertar una línea en blanco al final del fichero.

### Carga de datos.

Ubicaremos el fichero de 'CodigosPostales.txt' en un directorio conocido, y ejecutaremos el siguiente comando (adaptándolo al sistema operativo concreto que utilicemos):

```
COPY codigos_postales
(CPRO, CMUN, Nombre_Municipio, CP, Municipio_CP, Lugar_CP)
FROM 'c:/CodigosPostales.txt' WITH DELIMITER '\t';
```

Se observa que se han indicado los nombres de los campos. En el caso de que existiera el mismo número de campos en el fichero de texto que en la tabla, se podría omitir sus nombres.

### Consultas de prueba.

Para verificar si se han cargado los 10.144 códigos postales que contiene el fichero de texto, realizaremos la consulta:

```
SELECT count(*) FROM CODIGOS_POSTALES;
```

Para verificar si existe una ciudad concreta podemos realizar la consulta:

```
SELECT * FROM CODIGOS_POSTALES
WHERE NOMBRE_MUNICIPIO LIKE '%ADRID%'
ORDER BY CP
LIMIT 1;
```

(es interesante repetir la SELECT, eliminando la instrucción LIMIT 1)